1 Lab+Hwk 2: Coordinated and collective movements in Webots

This laboratory requires the following equipment:

- C programming tools (gcc, make).
- Webots simulation software.
- Webots User Guide
- Webots Reference Manual.

The laboratory duration is about 2 hours. Homework will be due on **the 6th day after your lab session, at 5 p.m**. Please copy your solutions, plots, descriptions (*.doc, *.ps, *.html, *.pdf, *.txt), movies, and so on into the directory ~ *correll/lab02/yourusername/*. The directory ~ *correll/lab02* allows you to write but not to read. The corresponding Webots files (see section 2 for details) should be saved in a sub- directory *webots/*. Please strictly follow these guidelines to facilitate the grading process, and keep in mind that no late solutions will be accepted unless motivated by health reasons (doctor certificate required).

1.1 Grading

In the following text you will find several exercises and questions. The notation $\mathbf{I_x}$ means that the problem has to be solved by implementing a piece of code and performing a simulation. The notation $\mathbf{S_x}$ means that the question can be solved using only additional simulation. The notation $\mathbf{Q_x}$ means that the question can be answered theoretically without any simulation. **Please use this notation in your answer file**!

The number of points for each exercise is given between parenthesis. The total sum of the points is for the laboratory and homework part is 100.

2 First Part: Simple Collective Motion

In the first lab, you programmed an obstacle avoidance behavior using the distance sensors of the Khepera. In *~correll/webots/* you find the worlds *flocking1.wbt, flocking2.wbt,* and *flocking3.wbt,* and controllers *leader, leader3, follower1, follower2, follower3,* and *flock_super.* In all these worlds, there will be both leader and follower robots. Leader robots are red in color, while followers are green. Copy all worlds and the controllers into your *webots* directory.

The controller *leader.cc* enables you to control a robot using the arrow keys on your keyboard. The goal of the first part of this exercise is to implement a simple following behavior using the Braitenberg vehicle framework in *follower1.cc* that allows a follower robot to follow a leader robot.

Remark: In order to facilitate the grading of exercise set, copy all your **controller**, **supervisor**, and **world** modules to *~correll/lab02/youruser-name/webots/worlds* and *~correll/lab02/yourusername/webots/controllers*

only after you finish the exercise. World files contain all the information not only for recovering the environment but also the corresponding controllers and supervisor programs: **make sure that you save them and deliver as part of your solution even if not explicitly mentioned**! Answer to the theoretical questions, a description of where the files can be found and their names, plots, movies[1], and so on should be saved instead in the directory *~correll/lab02/yourusername*. Do **not** save your controller, supervisor, and world modules in this directory but use the appropriate subdirectories!

2.1 Communication

In this lab, certain information must be transferred between robots. This could be accomplished using wireless radio transmissions or by broadcasting data with infrared. Robots that need to communicate have two additional nodes, namely an emitter and a receiver node, allowing for emitting and receiving data over a fixed range and a dedicated, arbitrary channel[2]. This will be used to allow robots to track wheel speeds (part 3) and relative positions (part 4) of other robots.

2.2 Flocking Rules

In "Flocks, Herds, and Schools: A Distributed Behavioral Model, in Computer Graphics" by Reynolds, a successful simulation of flocking is described. Reynolds lists several rules which he determines are needed for realistic flocking. They are:

1. Collision Avoidance: avoid collisions with nearby flockmates
2. Velocity Matching: attempt to match velocity with nearby flockmates
3. Flock Centering: attempt to stay close to nearby flockmates

In this lab, you will explore the influence and importance of each of these rules.

2.3 Implementing a following behavior

The most primitive type of flocking is one robot (follower) following another (leader). This can be accomplished using Braitenberg vehicles. You will now modify the Braitenberg obstacle avoidance routine to accomplish this behavior.

I_1 (10): Load *flocking1.wbt*. The leader robot is controlled by the *leader* controller to provide manual control. Modify the *follower1* controller to obtain a Braitenberg vehicle able to follow the leader robot. Save your solution as *follower1.cc*. Hint: Have a look at the Braitenberg vehicles that implement light following and avoidance from last exercise; what is the difference between them?

[2] Both range and channel are parameters that can be set in the Scene Tree Window. The channel can also be changed dynamically by the robot itself.

3 Second Part: Formation Control

For some applications (e.g., lawn-mowing, vacuum cleaning) it might be useful for having your flock maintain a relative formation (i.e. if the robots form a line, and the leader turns, the line itself turns). Figure 1 shows an example of this type of behavior. You will now implement relative formation flocking, using robot wheel speeds.

Q_2 (10): Compute the left and right wheel speeds of robot 2 (v_{l2}, v_{r2}) as a function of robot 1 (v_{l1}, v_{r1}), shown in *Figure 2-1*, as they perform a translation of a length T and a rotation in formation of radius R while maintaining the distance d between them. Hint: use the following formulas

$$\begin{aligned} R_{r1} &= R_{l1} + axle_length \\ \frac{v_{r1}}{R_{r1}} &= \frac{v_{l1}}{R_{l1}} \end{aligned}$$

with $R_{\ln}$ and R_{rn} being the radius of the left and right wheel of robot n, respectively, and $axle_length$ being the distance between the wheels of a single robot. With these formulas, you can calculate R_{l1}, R_{l1} and some term

$$\alpha = \frac{v_{xn}}{R_{xn}}$$

which is constant for any velocity/radius pair. This allows you to calculate the wheel speeds of the follower robot as a function of their radius with

$$v_x = \alpha R_x$$

I_3(5): Implement formation flocking behavior using *flocking2.wbt, leader.cc,* and *follower2.cc,* and save it as *follower2.cc*. In this situation, robot 2 can observe the wheel speeds of robot 1, and robot 3 can observe the wheel speeds of robot 2 (robot 3 should use the same equations as robot 2 by symmetry). You should initially just have the leader travel in a circle, with no obstacles, but may try more complicated formations if you wish. Take note of how the system performs.

S_4 (*): Try changing the leader speed to different values. Note what happens when turning if you set the leader speed to a very high value. (Points for this problem will be assigned based on the answer to the corresponding question in the homework).

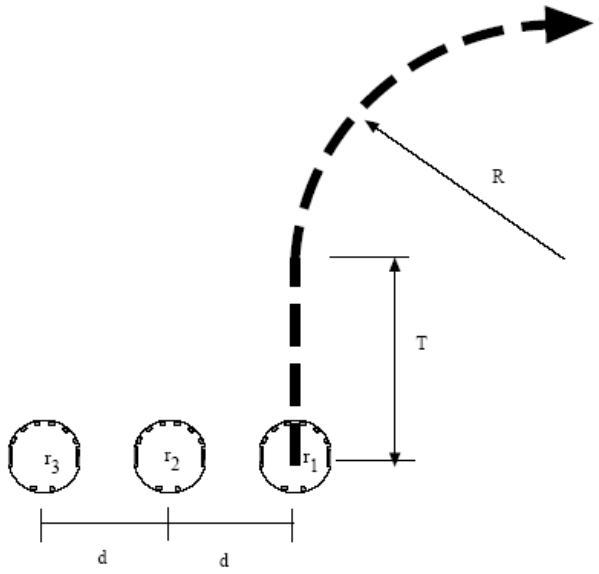

Figure 1: The robot formation first translates a length T and than rotates along a circle of radius R. d represents the inter-robot separation that should be maintained during the movement.

4 Third Part: Robust Flocking

In this part, you will attempt to implement flocking using the relative positions of robots. Copy the *flocking3.wbt* world and *leader3*, *follower3*, and *flock_super* controllers to your home directory (*flock_super* is the controller for the world supervisor[3]). The robots begin in a diamond formation with the leader at the head. The leader will travel a random path around the arena. Followers have the leader robot's relative range (distance between leader and follower) and bearing (relative angular offset of the leader's location in radians, i.e. 0 is straight ahead, $\pi/2$ is directly right, $-\pi/2$ is directly left) available to them.

I_5 (30): Implement a flocking algorithm for the follower robots so that the diamond formation is roughly maintained, using the relative leader position and proximity sensors. You must still use obstacle avoidance. You should use the same forward bias speed for all the follower robots. Save your code as *follower3.cc*.

S_6 (10): Test the robustness of the controller you implemented in question **I_5** with different robot bias forward speeds. Randomly assign different bias speeds to each robot within an interval $\pm \Delta$ and quantitatively determine the threshold Δ where your control schema breaks. Higher thresholds will garner more points, but will also be more vigorously verified.

5 Homework

Q_7 (5): Recall the performance of your following controller in **I_1**. How robust is your new controller when the first robot tries to avoid a wall? How constant

[3] Supervisor is useful agent in Webots. See the User Manual for more information on supervisors.

is the distance between both robots? What happens to the effective forward speed of the first robot when the follower is introduced?

Q_8 (5): Recall the performance of your flocking controller in **I_3**. What did you observe? How does avoiding obstacles affect the performance of the controller?

Q_9 (5): What happened when you increased the speed of the leader in **S_4**? Why did this occur? Does this reflect what would occur on real robots, or is it just an aspect of the simulation environment?

Q_{10} (5): How does the flocking capabilities of robots that can track the leader with the same bias speeds compare to the previous flocking strategy? What about with different bias speeds? What does this tell you about the importance of Reynold's different rules?

Q_{11} (5): What would happen if we maintained the same type of system with a designated leader robot, and increased the number of followers? Recall that the communication abilities will only work for a finite range. Can you think of a way which the system could be changed to work better in this situation?

Q_{12} (5): In this program, although they are given the information, we don't give a physical method by which the follower robots track the relative position of the leader robot. What are some ways that this could be accomplished with real robots? What might be the pros and cons of these strategies? Recall that robots do NOT have access to their global coordinates.